



Sandia National Laboratories

A COMPARISON OF PRECONDITIONERS FOR SOLVING LINEAR SYSTEMS ARISING FROM GRAPH LAPLACIANS*

KEVIN DEWEESE[†] AND ERIK G. BOMAN[‡]

Abstract. We consider the solution of linear systems corresponding to the combinatorial and normalized graph Laplacians of large unstructured networks. We only consider undirected graphs, so the corresponding matrices are symmetric. A promising approach to solving these problems is to use a class of support graph preconditioners. We previously implemented such a preconditioner in Trilinos in serial using the Epetra software stack. This work extends that implementation to run in parallel on distributed memory systems and migrates the implementation to the Tpetra software stack to help with future development. This preconditioner is compared against other preconditioners currently available in Trilinos. We show that domain decomposition is an effective preconditioning method for network problems. Our support graph preconditioner can be used as a local (serial) subdomain solver.

1. Introduction. Networks play an important role in many application areas, for example engineering, social sciences, and biology [8]. We focus on networks that are large and unstructured. Several analysis techniques rely on solving linear systems and eigensystems of graph Laplacians such as random walks [4] and Katz centrality scores [6] using linear solvers and graph partitioning [10] and clustering [11] using eigensolvers. These solvers can be very compute intensive tasks but preconditioners can be used to dramatically reduce the solution time. Although there are many good preconditioners for PDEs, they are generally not well suited for graph Laplacians from highly irregular graphs or scale-free networks. Several graph based preconditioners with strong theoretical results have been developed over the past decade [7, 9]. However these are difficult to implement. We instead seek to better understand the support tree preconditioner first proposed by Vaidya [1] and how it compares to other preconditioners on these graph problems.

1.1. Background. The combinatorial Laplacian of an undirected graph G is given by

$$L_G = D - A_G$$

where A_G is the adjacency matrix of G and D is the diagonal matrix containing the sum of adjacent edge weights, or in the unweighted case just the vertex degree. A special scaling of this matrix called the normalized Laplacian is given by

$$N_G = D^{-1/2} L_G D^{-1/2}.$$

Both L_G and N_G are positive-semidefinite and diagonally dominant. An interesting note is that solving $L_G x = b$ can be done indirectly by solving $N_G x' = b'$. We can see this by setting $x' = D^{1/2} x$ and $b' = D^{-1/2} b$ yielding the following.

*This paper will be included in the CSRI Summer Proceedings 2013 from Sandia National Laboratories. It is also available as technical report SAND-2013-9772-P.

[†]UC Santa Barbara Dept. of Computer Science, kdeweese@cs.ucsb.edu. This work was performed during an internship at Sandia.

[‡]Sandia National Laboratories, egboman@sandia.gov. Sandia is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000.

$$N_G x' = b'$$

$$D^{-1/2} L_G D^{-1/2} D^{1/2} x = D^{-1/2} b$$

$$D^{1/2} D^{-1/2} L_G x = b$$

$$L_G x = b$$

Thus if solving one system is more efficient than the other, perhaps the normalized Laplacian as it is typically better conditioned, then it could be solved and the solution converted. However the eigenproblems must be solved separately as each Laplacian has a different spectra.

1.2. Preconditioners. A good preconditioner M for a matrix A should reduce the number of iterations of the preconditioned system $M^{-1}A$. In other words M should be a good inverse approximation of A . In addition solving the system $Mw = y$ should be much easier to solve than $Ax = b$ as it will be solved at every iteration. The first constraint prompts us to bound the condition number of $M^{-1}A$ while the second constraint requires us to bound the fill in the triangular factors of the preconditioner. Assuming a complete Cholesky factorization is used these factors will be of the form $M = CC^T$ and will be used to quickly solve $CC^T w = y$. Perhaps the most simple preconditioner is the Jacobi method that sets $M = D$ where D is the diagonal of A . While not a very good inverse approximation this preconditioner is very cheap to apply at every iteration. A slightly better inverse approximation could be used such as the symmetric Gauss-Seidel preconditioner that decomposes the input matrix into triangular parts $A = L + D + U$ and uses $M = (D + L)D^{-1}(D + U)$. Another popular preconditioning technique is to use an incomplete Cholesky factorization to approximately factor the matrix $A \simeq M = \tilde{C}\tilde{C}^T$ by dropping some entries during the factorization of A .

The first of a class of support graph preconditioners was proposed by Vaidya. The basic version finds a maximum-weight spanning tree of the graph of A and uses this as a preconditioner. (For details, see [1].) This preconditioner has condition number $O(nm)$ which bounds the number of iterations of the preconditioned system to $O(\sqrt{nm})$, where n is the dimension of the matrix (number of vertices) and m is the number of non-zero entries of the matrix (number of edges). However these are worst case bounds and typically the number of iterations required is much less. Since the preconditioner corresponds to a tree it can be factored with no fill.

2. Software. A version of Vaidya's preconditioner was previously implemented inside the Ifpack package of Trilinos [5]. Ifpack uses Epetra linear algebra primitives as opposed to the more recent, templated Tpetra primitives. Tpetra allows arbitrary scalar data types and arbitrary index types. Most future Trilinos development will focus on packages using this new Tpetra stack. For these reasons we decided to migrate all future work concerning support graph preconditioners to this new Tpetra software stack. The packages relevant to our work can be seen in Figure 2.1.

To accomplish this migration and improve upon the previous support graph implementation a few pieces of software were added to the Trilinos software library.

Feature	Old Stack	New Stack
Core	Epetra	Tpetra
Sparse Direct	Amesos	Amesos2
Preconditioners	Ifpack	Ifpack2
Partitioning and Ordering	Zoltan	Zoltan2

FIG. 2.1. *Trilinos software stack*

Previously Trilinos did not contain a sparse Cholesky solver. Using the Amesos2 adapter package an interface was added to the CHOLMOD package [2]. Additionally a support graph preconditioner was added to the Ifpack2 preconditioner package which creates the support graph and calls the CHOLMOD interface to perform a complete factorization. Furthermore a bug in Ifpack2's Additive Schwarz domain decomposition class was corrected so that it could be used with the support graph preconditioner.

3. Experimental. Experiments were run to solve preconditioned linear systems using the conjugate gradient solver in the Belos linear solver package in Trilinos. The right hand sides were generated by multiplying L by a random solution x . Experiments were run on 4 different graphs from the University of Florida sparse matrix collection [3] shown in Figure 3.1. The first 3 graphs in this table are network graphs and experiments were run using both the combinatorial Laplacian and normalized Laplacian matrices of these graphs. An important note is that these graphs are all unweighted so the support graph preconditioner of the combinatorial Laplacian is simply a random spanning tree. The last graph is a stiffness matrix problem included to see how a support graph preconditioner fares on a more traditional problem. Experiments with the MSF support graph preconditioner were run with slightly random edge weights so that the on the unweighted combinatorial Laplacian a random tree would be selected every time. The diagonal of the original matrix was kept for the preconditioner and on the F1 graph was scaled so the preconditioner would be positive-definite. The performance of the support graph preconditioner was compared against other Ifpack2 preconditioners. These include Jacobi, Symmetric Gauss-Seidel (SGS), and Incomplete Lower-Upper (ILUT). In parallel, Ifpack2's Additive Schwarz with no overlap was used with the support graph and ILUT preconditioners as sub-domain solvers. Ideally ILUT would be replaced with Incomplete Cholesky as we are dealing with symmetric matrices but this is currently not implemented in Ifpack2. The default ILUT parameters were used including a drop tolerance of 10^{-12} and fill value of 1. Additionally ILUT required a small, relative scaling of the diagonal (~ 1.01) to ensure a positive-definite preconditioner. All experiments were run on a 64 core shared-memory linux server (vesper@sandia.gov). Zoltan2's interface to the Scotch partitioner was used to distribute matrix rows amongst processors. Experiments were done with partitioning turned on and off using 8 cores to demonstrate the effect of partitioning on the various solvers. Scaling experiments were performed up to 32 cores.

4. Results.

4.1. Serial. The results of serial solves on the flickr graph for both the combinatorial and normalized Laplacians can be seen in Figure 4.1. It is clear from using no preconditioning on the combinatorial Laplacian that some preconditioning method is needed. In the combinatorial case MSF yields the best solution time with SGS

Graph	Rows (Vertices)	NNZ (Edges \times 2)
flickr	820,878	13,250,560
as-Skitter	1,696,415	22,190,596
hollywood-2009	1,139,905	57,515,616
F1	343,791	26,493,322

FIG. 3.1. *Graphs used in experiments*

	Iters.	Solve Time (s)		Iters.	Solve Time (s)
MSF	42	6.247	MSF	54	7.713
Jacobi	76	7.841	Jacobi	86	8.942
SGS	25	7.411	SGS	49	14.32
ILUT	30	8.331	ILUT	64	15.85
None	2689	266.3	None	86	8.573

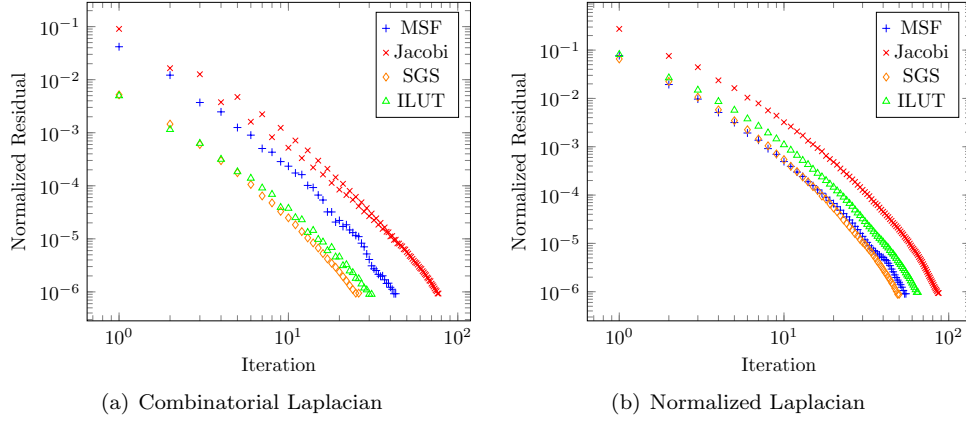
(a) Combinatorial Laplacian
(b) Normalized Laplacian

FIG. 4.1. *Serial results on flickr graph*

yielding the fewest number of iterations. The normalized Laplacian is scaled so that solving without preconditioning is equivalent to using Jacobi. Interestingly, in the normalized case the preconditioned solves are all slightly more expensive. One idea being considered to solve the combinatorial problem involves solving the normalized problem and converting the solution vector since the normalized problem should be better conditioned. However, these results seem to suggest such a method would not be fruitful. In Figure 4.2 the convergence rates are shown by the normalized residual at each iteration. Interestingly, the preconditioners seem to have very similar convergence rates that just appear to be off by some constant which seems to be smaller in the normalized case.

4.2. Partitioning. The Scotch graph partitioning algorithm was used in parallel solves to try and increase the quality of the preconditioner and improve the load balance across processors. The results with partitioning turned on and off for 8 processors on the as-Skitter combinatorial Laplacian are shown in Figure 4.3. Improvement in the quality of the preconditioner can be inferred by the change in iteration count while the change in run time is some mix of load balance improvement and change in preconditioner quality. Jacobi behaves as expected; since it just uses diagonal scaling the quality of the preconditioner does not change but load balancing greatly improves performance. Only with ILUT does using partitioning seem to greatly improve the quality of the preconditioner. Each sub-domain incomplete factorization needs to have as many edges as possible to improve quality. However, the number of edges in the MSF sub-domain is constant so having an extra edge in the sub-domain won't change the quality much. The graph algorithm might have selected this edge instead but some other edge would not have been selected. This leads us to believe that the MSF preconditioner should scale reasonably well since losing the edges off processor won't hurt the preconditioner quality much. Since turning partitioning on always leads to at least slightly better performance due to better load balancing, it is always turned on for the scaling experiments in the next section.

4.3. Scaling. Up to 32 cores were used to perform scaling experiments on as-Skitter and hollywood-2009. Results for as-Skitter's combinatorial Laplacian are

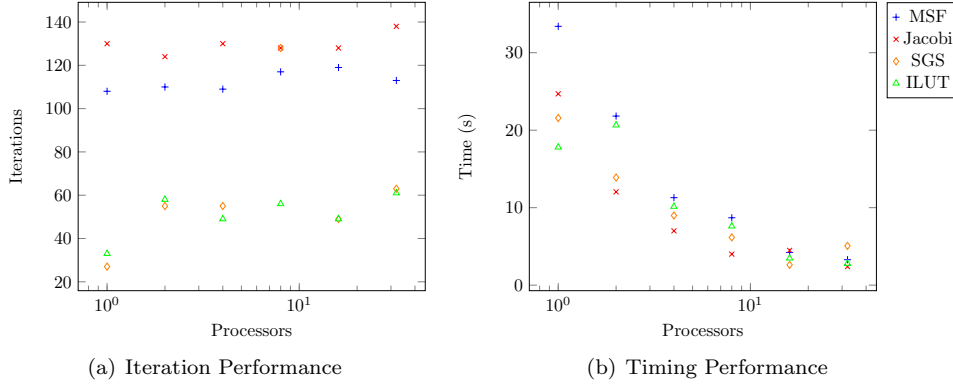
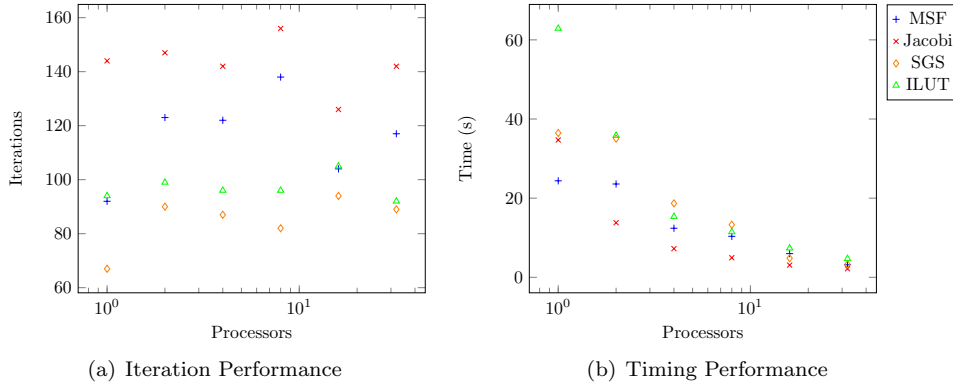
FIG. 4.2. *Iterations to convergence*

	Iters.	Solve Time (s)		Iters.	Solve Time (s)
None	18	9.516	None	122	10.71
Scotch	17	8.695	Scotch	128	4.005
(a) MSF			(b) Jacobi		
	Iters.	Solve Time (s)		Iters.	Solve Time (s)
None	116	16.39	None	120	16.45
Scotch	128	6.166	Scotch	56	7.613
(c) SGS			(d) ILUT		

FIG. 4.3. *Partitioning results using 8 cores on as-Skitter combinatorial Laplacian*

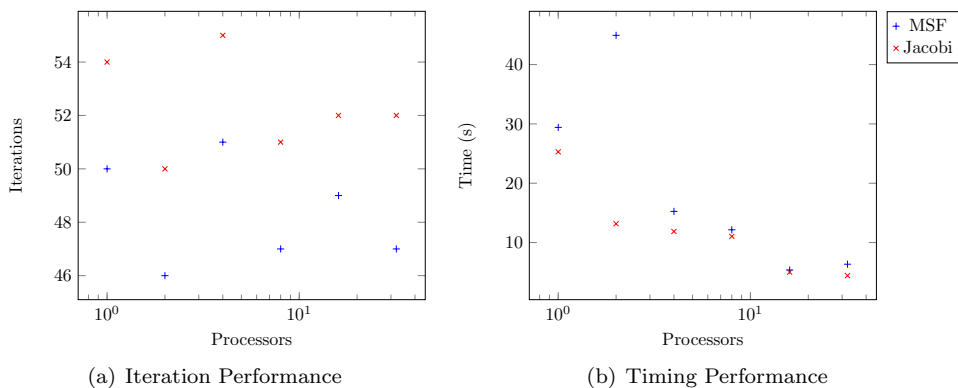
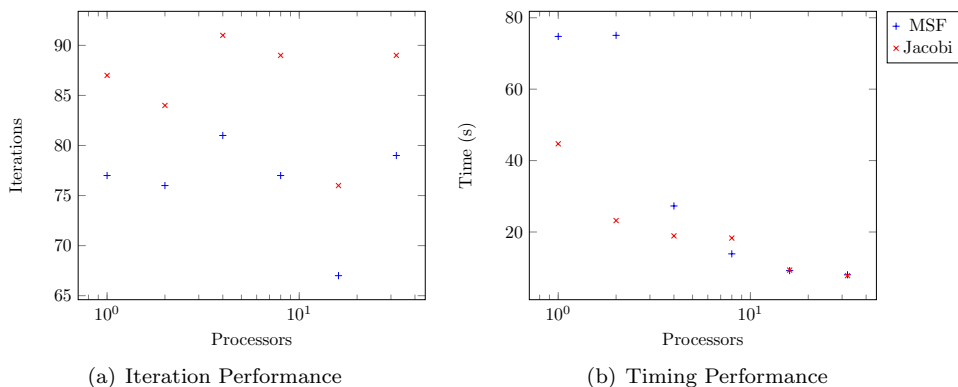
shown in Figure 4.4. A trial with no preconditioning was run but the performance was so poor (10 to $100\times$ slower) that the results are excluded. Results for as-Skitter's normalized Laplacian are shown in Figure 4.5. The number of iterations required for each preconditioner doesn't fluctuate much as the number of processors increases and their order stays relative the same. Some fluctuation in iteration count is expected due to using a random right hand side during every solve. Jacobi seems to have the best solve time performance though as the number of processors increase the gap between the preconditioners decreases. Since Jacobi and MSF seem the most competitive the experiments on the larger hollywood-2009 were run only with them. The combinatorial results are shown in Figure 4.6 and the normalized results are shown in Figure 4.7. The number of iterations for both of these methods seems to fluctuate a bit more on this larger graph though the gap between them stays the same. With the exception of MSF on 2 processors both methods seem to scale well regarding solve time. We suspect that there is some initial MPI overhead causing the spike at 2 processors which is quickly overcome.

4.4. Stiffness Matrix. Out of curiosity we ran one experiment on a more traditional stiffness matrix F1. This matrix is symmetric positive-definite so no modifications were needed to use Belos' CG solver. However, this matrix has both positive and negative off-diagonal entries so the diagonal of the MSF and ILUT preconditioners had to be modified to ensure the preconditioners were positive-definite. Solves were done using 32 cores and the results are shown in 4.8. ILUT and MSF perform

FIG. 4.4. *Scaling on as-Skitter graph combinatorial Laplacian*FIG. 4.5. *Scaling on as-Skitter graph normalized Laplacian*

relatively better against Jacobi and SGS than they did in with the network graphs. The structure of this graph is much simpler so we suspect that the main reason is the introduction of edge weights.

5. Conclusions. It is clear that some method of preconditioning is required to solve linear systems coming from graphs. However, it is not clear if a support graph preconditioner is useful or if something as simple as Jacobi should be used instead. We observed that Jacobi preconditioning works quite well: even if the iteration count is high, each iteration is very fast. We remark that MSF is a simple support graph preconditioner and there is potentially room for improvement by choosing better sub-graphs. Domain decomposition was shown to be a good parallel preconditioner as the number of iterations stayed almost constant as the number of processors (subdomains) increased. Partitioning has been shown to be useful for all the preconditioners tested but most important for SGS and ILUT. All of the preconditioners seem to scale well on the graphs used as the number of processors increase. This suggests that using additive Schwarz with local (serial) preconditioning is a viable approach for network problems. We observed that it is typically more difficult (longer run times) to solve for the normalized Laplacian so using the normalized Laplacian to solve a system arising from the combinatorial Laplacian does not seem viable.

FIG. 4.6. *Scaling on hollywood-2009 graph combinatorial Laplacian*FIG. 4.7. *Scaling on hollywood-2009 graph normalized Laplacian*

6. Future Work. Most of the software for these experiments was written recently and has had limited testing. Tests and documentation are needed before this software is brought out of experimental status. There are a few followup experiments that should be run. A set of weak scaling experiments would help understand how these methods perform as graph sizes increase. This will require choosing a reasonable graph generator. These experiments used the matrix ordering in the original files and it is possible that this ordering might be helping some of the methods. In particular, the incomplete Cholesky/LU and Gauss-Seidel methods are known to be sensitive to ordering. Therefore, an investigation of the effect of ordering should be performed.

Acknowledgment.

The authors would like to thank Karen Devine, Rich Lehoucq, and Siva Rajamanickam.

REFERENCES

- [1] M. BERN, J. R. GILBERT, B. HENDRICKSON, N. NGUYEN, AND S. TOLEDO, *Support-graph preconditioners*, SIAM J. on Matrix Anal. and Appl., 27 (2006), pp. 930–951.
- [2] Y. CHEN, T. A. DAVIS, W. W. HAGER, AND S. RAJAMANICKAM, *Algorithm 887: Cholmod, supernodal sparse cholesky factorization and update/downdate*, ACM Trans. Math. Softw.,

	Iters.	Solve Time (s)
MSF	408	3.106
Jacobi	580	4.254
SGS	297	7.168
ILUT	199	3.654

FIG. 4.8. *F1 stiffness matrix solved using 32 cores*

- 35 (2008), pp. 22:1–22:14.
- [3] T. A. DAVIS AND Y. HU, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1:1–1:25.
 - [4] L. GRADY, *Random walks for image segmentation*, IEEE Trans. Pattern Anal. Mach. Intell., 28 (2006), pp. 1768–1783.
 - [5] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, T. G. KOLDA, R. . B. LEHOUCQ, K. R. LONG, R. P. PAWLOWSKI, E. T. PHIPPS, A. G. SALINGER, H. K. THORNQUIST, R. S. TUMINARO, J. M. . WILLENBRING, A. WILLIAMS, AND K. S. STANLEY, *An overview of the trilinos project*, ACM Trans. Math. Softw., 31 (2005), pp. 397–423.
 - [6] L. KATZ, *A new status index derived from sociometric analysis*, Psychometrika, 18 (1953), pp. 39–43.
 - [7] I. KOUTIS, G. MILLER, AND R. PENG, *Approaching optimality for solving SDD linear systems*, in Proc. of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, IEEE Computer Society, 2010, pp. 235–244.
 - [8] M. NEWMAN, *Networks: An Introduction*, Oxford University Press, Inc., New York, NY, USA, 2010.
 - [9] D. A. SPIELMAN AND S.-H. TENG, *Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems*, in Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, STOC '04, New York, NY, USA, 2004, ACM, pp. 81–90.
 - [10] D. A. SPIELMAN AND S.-H. TENG, *Spectral partitioning works: Planar graphs and finite element meshes*, Linear Algebra and its Applications, 421 (2007), pp. 284 – 305.
 - [11] U. VON LUXBURG, *A tutorial on spectral clustering*, CoRR, abs/0711.0189 (2007).